styra

Styra eBook

OPA at Scale

Managing consistent, secure and scalable rules across teams, clusters and clouds

In this ebook, we lay out the key drivers for managing policy and authorization at scale across the cloud-native stack.

Specifically:

- Establishing a standard building block for policy and authorization
  - Leveraging Open Policy Agent (OPA)
- Scaling OPA: creating an environment that makes seamless, rapid collaboration and automation possible
- Managing OPA at Scale: empowering enterprise-wide teams to take advantage of policy-as-a-code for unified policy and authorization

Introduction

With the rapid growth of cloud-native services and containerized applications, enterprises have been facing a quiet, universal and mounting crisis: there has been no way to create and deploy consistent, secure and scalable rules across the cloud-native stack. This is not an insignificant challenge, as these rules, known as policy and authorization, are at the foundation of building secure and scalable applications — the core differentiators of cloud-native business.

But compared to traditional, monolithic application environments, managing these rules in the cloud is a much more difficult challenge — by orders of magnitude. Thus far, organizations have had no good way to manage this complexity and chaos at scale.

The challenge is that the cloud is something like the "Wild West" — ungoverned by any single standard — when it comes to policy and authorization, or the "who can do what" and "what can do what" rules that make security and scalability possible. While organizations desperately need a comprehensive standard that spans every cloud-native environment, such rules are instead maddeningly unique to every organization, development team and microservice — hand-crafted, in different programming languages, with bespoke logics. It seems like a contradiction that these policies must, at the same time, scale uniformly across the entire cloud-native stack — from Kubernetes, to microservices, to application authorization, to infrastructure management, to the CI/CD pipeline to much more.

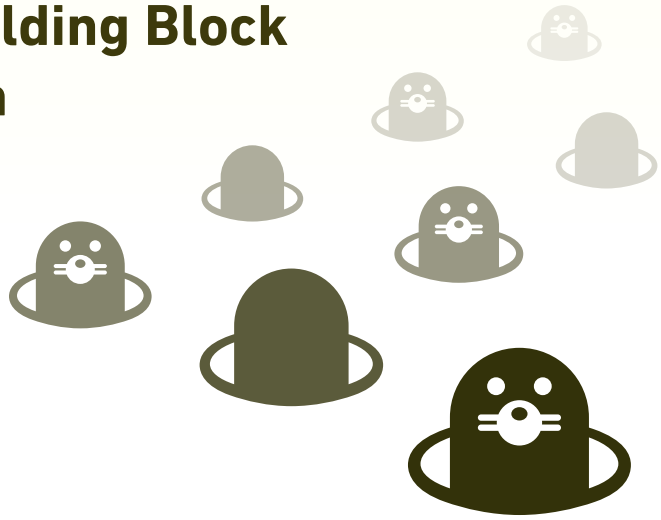
Organizations are aligning around a common vision about how to unify policy and authorization across all cloud-native environments. Specifically, they're looking at how to:

+ Modernize how they create, validate and enforce cloud-native rules across teams, services and environments
+ Streamline and scale production
+ Define rules and validate changes for security and compliance
+ Manage chaos and complexity in a uniform and consistent manner
+ Enable seamless collaboration across the enterprise

And scale these policies must. Indeed, organizations are learning they must adopt a consistent policy framework across the entire cloud. By "solving" this authorization problem, organizations can immediately begin to improve security and compliance, mitigate human error and misconfigurations, accelerate application development and reduce the "mental tax" of policy creation. Moreover, wth a standard policy framework in place, developers can avoid continually reinventing the wheel — and instead spend their time on revenue-driving tasks like building better applications that differentiate the business.

Chapters

Establishing a Standard Building Block for Policy and Authorization



Manually "doing security" in today's cloud-native environments is like a never-ending game of whack-a-mole where you can see only a fraction of the moles. The reason? Legacy policy and authorization is centered around people, accounts and roles — who can access what. Compounding this issue is the fact that traditional monolithic applications typically have authorization systems hard-coded into the apps themselves.

In order to meet the requirements of today's dynamic, containerized application environments, policy and authz must extend to control services, software-defined infrastructure and application code itself. We must control not just who, but what can do what to build, maintain and prove security and compliance.

But in creating, deploying and managing such policies, enterprises face a number of thorny, interrelated challenges. The first is that **software development has accelerated dramatically.** Indeed, enterprises have moved from updating software monthly or quarterly, to dozens of times per day. At the same time, **developers and developer teams are expected to manage policy and authorization themselves** — including security policies — even as developers are asked to prioritize time-to-market above all else. The result, on the one hand, is that **there are as many policies— each using a different coding language, tooling, and logic — as there are development teams.** Such disparate policies work very poorly together at scale, are difficult to analyze for security and compliance, and quickly become impossible to manage. On the other hand, **developers are always security experts,** which means that in the rush to deploy code, security often gets short shrift.

The way to solve these challenges is to provide developers with better tools for creating and managing policies at scale, and for shifting security left — providing every development team with consistent, validated security policy, in other words (it also becomes difficult to port skills, tools, people and processes between teams).

At the same time, enterprises have moved from updating software monthly or quarterly, to dozens of times per day. When developers are asked to prioritize time-to-market above all else, this means that security policies, which are also managed by developers today, often get short shrift. This means developers need better tools for policy building and to shift security left, so that speed and security are no longer at odds in a software-defined world. This also means that the traditional way of managing policy and authorization is simply unworkable.

The step on that journey is to adopt a standard building block for managing cloud-native policy and authorization. This standard must be understandable across not only development teams, but by every technology in your cloud-native production environment. In the same breath, you also need to provide every development team with validated, secure, performant, fine-grained policy controls. This ensures that you can have not only consistent security and compliance, but also that security policies and skills are portable across teams, services, clusters, clouds — your entire organization.

Leveraging Open Policy Agent (OPA) - the open-source, do-it-all policy engine

In light of these challenges, many companies are turning to the open-source project Open Policy Agent (OPA), created by Styra, and now part of the Cloud Native Computing Foundation. A general-purpose policy engine, OPA lets organizations unify policy and authorization across the cloud-native stack.

OPA is a graduated project within the CNCF, sharing that distinction with the likes of Prometheus, Envoy, Helm and even Kubernetes itself. This means OPA is a mature project, with strong, active community, and proven deployments, in business-critical production environments for hundreds of cloud-native leaders, including companies like Netflix, Atlassian, Goldman Sachs and many more. OPA is domain-agnostic and highly flexible and extensible, which means that it works in

virtually any cloud-native environment, for any policy use case. As a result, OPA enjoys a rich off-the-shelf integration ecosystem and a thriving developer community, growing at a pace of over a million downloads a week. Its high-level declarative language lets you specify policy as code and, and through simple APIs, offloads policy decision-making from your software. This means that, with OPA, you can create consistent, secure and widely applicable policies across your entire cloud-native ecosystem. It also means that updating or changing policy requires no changes to application code — since OPA is decoupled and standalone. As a result, OPA has become the de facto standard for policy and authorization in the cloud.

Because of the popularity of OPA, many developers are at a stage where they're either experimenting with policies, testing out specific use cases or moving OPA into full production in business-critical environments.

So, how do you create an organization-wide standard for policy and authorization? Here are some tips for facilitating OPA adoption:

1. Build test policies to validate OPA in your environment.

2. Get your team on board with an OPA pilot project.

3. Build out a narrowly-defined OPA use case for production.

4. Expand your OPA policies and strategy to solve adjacent problems.

Use Case: Netflix microservices authorization

NETFLIX

Policy and authorization are developer-level challenges that have business-level implications. For instance, Netflix, an early pioneer of microservices, needed a way to manage service-level authorization for the dozens of services that comprise its back end — services that make Netflix video services possible. Put simply, Netflix needed a consistent policy toolset that each development team could use to control service-level access to the services they controlled, across numerous enforcement points, such as HTTP APIs, SSH and more. With such a complicated microserservices ecosystem where authorization decisions needed to happen in less than a millisecond, it was simply not possible for Netflix to massively scale its services without a robust policy standard; an amalgam of bespoke policies would introduce unreasonable latency and make the Netflix backend impossible to manage at scale.

As such, OPA was a natural fit for the Netflix environment. It allowed dozens of teams to standardize how they create and enforce policy with a consistent policy-as-code language, offload authorization decisions to the open-source policy engine and enable Netflix to make global changes to authorization policy at any moment across its back-end ecosystem.

Benefit to the company

The business benefits when every team leverages a unified framework for defining, validating and deploying policy and authorization. Not only does this save teams from constantly reinventing the wheel (just as no one has to reinvent container code, encryption standards, SSO protocols or MFA for authentication), it also reduces manual error while increasing productivity and automation. In other words, what was once a complex, time-intensive, impossibly manual project becomes a simple, streamlined process that improves both speed and security.

Who's doing it well:

Many leading enterprises use OPA in full, business-critical production, including Atlassian, Chef, Netflix, CapitalOne, Goldman Sachs, TripAdvisor, Pinterest, Intuit, T-Mobile and many more.

With a standard building block for policy and authorization in place, organizations can:

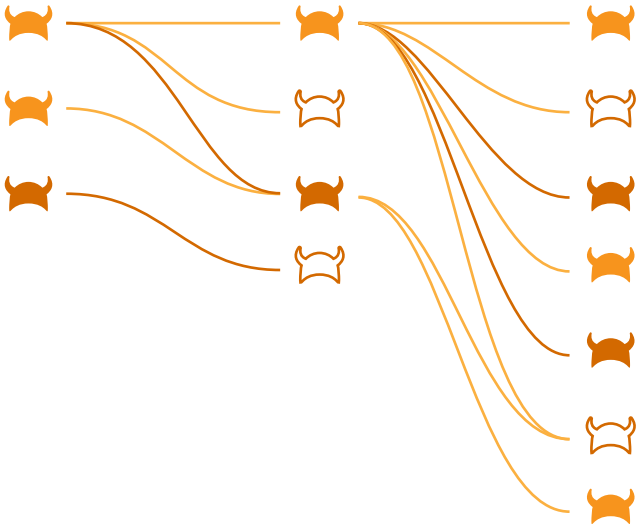
+ Quickly create and automatically enforce policy across dynamic multi-cloud and containerized environments.

+ Build security, compliance and operational errors directly into their processes and cloud infrastructure — helping to eliminate errors and close security gaps.

+ Express simple policies over complex, hierarchical data and bring order to the chaos.

+ Stop mistakes before they happen and focus on better building better apps, rather than on correction.

Scaling OPA: create an environment that makes it possible for collaboration to happen quickly and smoothly



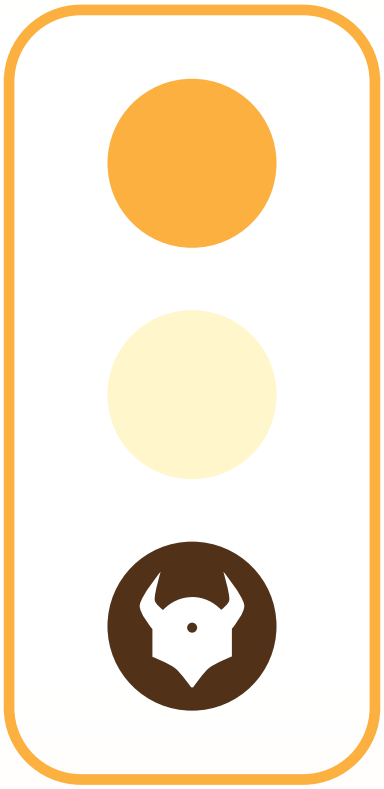
At a certain point, OPA users move from the sandbox and into production — and eventually to using OPA at true scale. Whether that means using OPA in a rapid-scaling microservices environment or across numerous services and teams, there are a number of considerations you will want to take to ensure seamless development and collaboration.

One thing to bear in mind: standards like OPA become more powerful the more widely they are used — just as with any standard, like Kubernetes, APIs or even HTTP. When many teams use a consistent, domain-agnostic approach to policy building, it enables your organization to establish well-vetted best practices, speed further adoption and automate policy processes for many developers.

Indeed, OPA users often find that once they invest in one use case, such as Kubernetes admission control or service mesh authorization, they can reuse much of their knowledge, tools and policies to solve authorization problems in adjacent areas. There is a "network effect" with OPA, and new services will fall into scope as your usage of OPA grows. This network effect also also makes it easier to port skills and processes across the organization. For instance, it becomes much easier to collaborate across times, move engineers to new projects, promote from within, and hire rockstars with key expertise when you use standardized tooling. At the same time, the risk of losing key players is significantly diminished, as policy systems don't rely on custom tribal knowledge to run.

Towards using OPA at scale, there are a number of best practices that can help your organization on its journey to create an environment for seamless, rapid collaboration — and managing secure, consistent policy everywhere.
Use this step-guide to for rolling out and standardizing OPA.

Rolling Out OPA and Policy-as-code



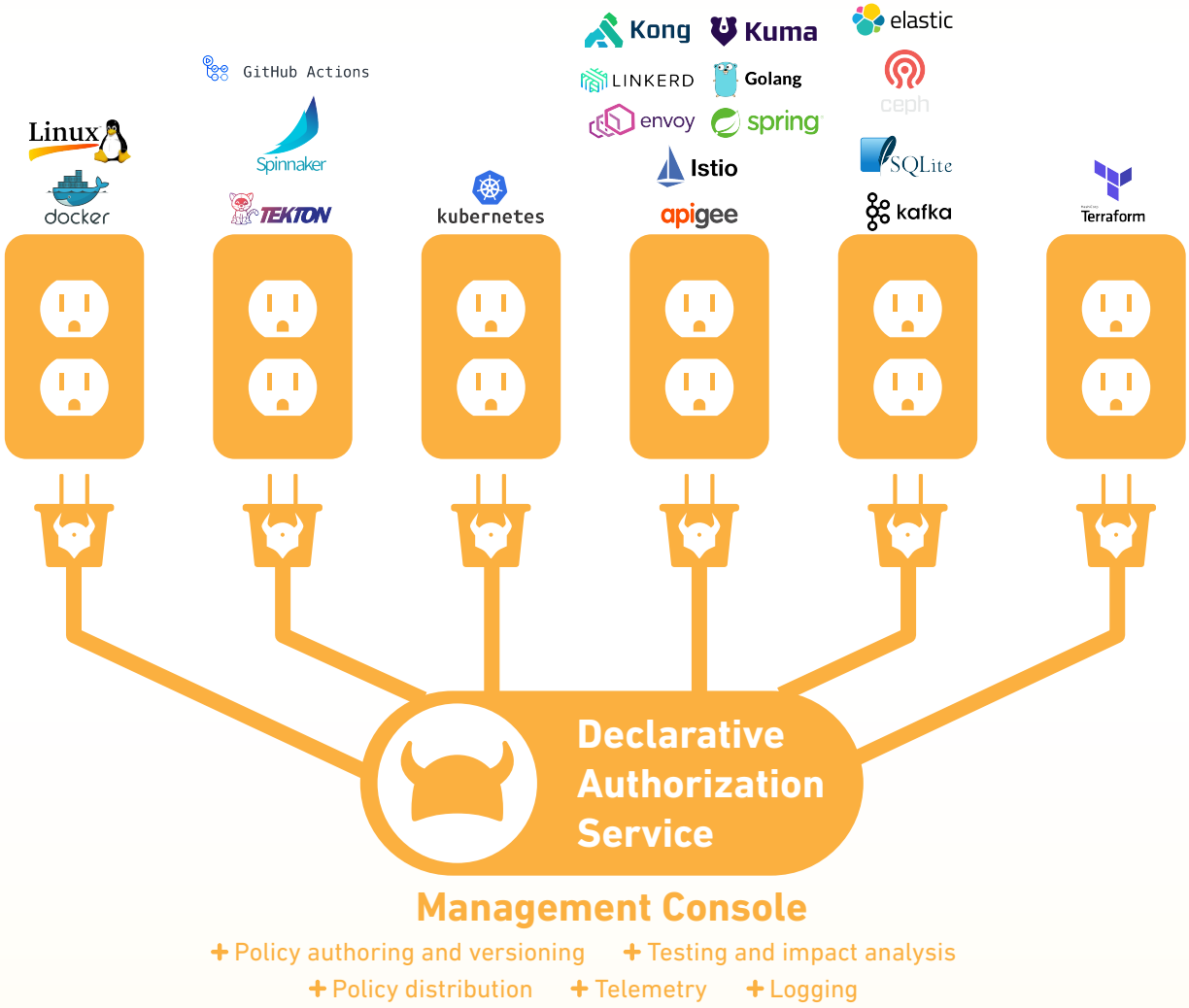
In rolling out OPA, we can distinguish between two stages: the Experimentation or Pre-Production Stage and the Production Stage. In combination, these stages take you from picking your OPA domain or use case (like Kubernetes or Envoy) and learning how to build policies, to deploying OPA in your environment, monitoring your deployment and logging decisions. These steps also enable you to not "just" learn OPA and Rego, its high-level declarative policy language, but treat policy-as-code just like all other code (as a first-class citizen).

As you work to deploy OPA, these are the boxes your organization should aim to tick.

Experimentation and Pre-Production Stage

Phase 1 Requirements (determine the playing field, ground rules and player data)

1. ***Choose your domain.*** Kubernetes? Envoy? Your CI/CD pipeline? Terraform? From the universe of tools across the cloud-native stack that integrate with OPA, pick a place to start for your project.



2. ***Assemble real-world policy.*** Often stored in wikis, PDFs, or in people's heads, these policies will translate into the policy-as-code you build with OPA. Clearly document this real-world policy in well organized spreadsheets so your team has a shared understanding.

3. ***Understand data dependencies.*** OPA works by making decisions about structured data (eg. JSON) — specifically, whether data conforms to or violates a policy. If you want to write a policy that ensures only certain types of users can access certain types of workloads, for instance, or that only PCI-compliant containers can talk to each other, you will need to feed OPA the data that defines which users and workloads have a role, or which containers are PCI-compliant. Work to understand where those parameters are defined.

4. ***Choose enforcement points.*** OPA decouples policy decision making from enforcement, meaning OPA decisions are handed off to be enforced by various APIs in your environment. In a microservices environment, that might be an Envoy proxy or sidecar authorization API; In Kubernetes, it might be an Admission Control validating webhook (if you want to control which containers can run in a cluster) or RBAC (if you want to control who or what can access that cluster).

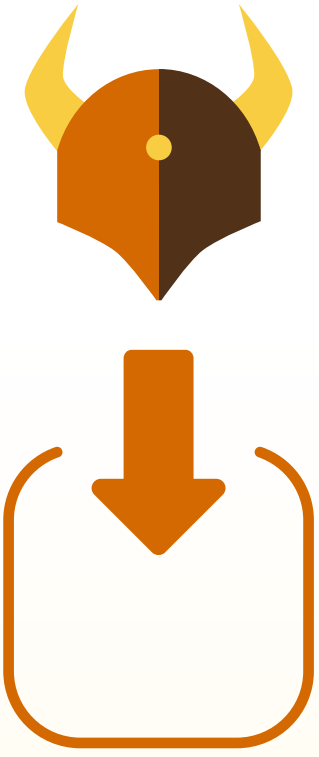
Phase 2 Enter OPA

1. ***Configure your services to use OPA.*** Whether you're writing policy for an Envoy proxy, Kubernetes Admission Control or a Kafka database, you will need to configure that service to talk to OPA. (Thanks to the OPA community, there's extensive documentation on this and the following points, including proven methods for more than 30 OPA integrations, see openpolicyagent.org.)

2. ***Deploy OPA.*** Simply download OPA.

3. ***Configure OPA.*** Ensure that OPA can communicate decisions to your services/APIs.

4. ***Harden OPA configuration.*** Lock down your OPA configuration, to ensure that OPA instances can only communicate with the appropriate Services/APIs.



Phase 3 Author Policy

1. ***Write and version control Rego.*** To create policy-as-code that's understandable by OPA, users will need to get a handle on Rego, OPA's high-level declarative policy language. Fortunately, there is comprehensive documentation about how to get started with, and master, Rego. Once you're familiar with Rego, you will need to version control your policies to ensure that they work as intended, just like you would with any other code in your application ecosystem. While learning Rego can seem like a challenge at first, remember that it enables teams to "learn once, and use many times" — because Rego is a single, unified policy language that allows your teams to write and deploy policies anywhere in your environment in the same way.

2. ***Decide and learn input and decision schemas.*** Once you understand your data dependencies, determine how to best feed that information to OPA, as structured JSON Data, so it can make an evaluation as quickly as possible.

3. ***Modularize policy and delegate for collaboration.*** There are many ways to configure OPA and its corresponding policies. Determine whether it makes more sense, for example, to maintain one large set of policy that iterates through a number of factors to make a decision, or if it would be better to maintain separate instances of OPA that each process one of those factors and communicate their decisions to the other OPA instances.

Production Stage

Successfully deploying OPA at enterprise scale entails more than just learning Rego; developers and platform engineers will also need to account for elements like the CICD pipeline, monitoring and logging. However, these stages will be familiar to any developer or platform engineer, as they are standard elements of any software lifecycle. Because OPA deals with policy-as-code, you can thus treat OPA policies like any other piece of code in your ecosystem — like a first-class citizen.

Phase 4 CI Stage

1. ***Assemble policies from different repositories.*** Make sure you have all of your policy-as-code in one place for streamlined deployment.

2. ***Test policies.*** Perform tests — both unit tests, and QA to validate that your policy-as-code works as expected. Tests are one of the best ways to lend OPA credibility in your organization (and put security and compliance teams at ease) — all a part of policy-as-code as a first-class citizen. Moreover, testing can help to codify developers' understanding of OPA and creates a safety net when creating new policies.

3. ***Create policy build artifacts (e.g. Bundles).*** Policy Bundles are the way that OPA policies are pushed through your build pipeline and deployed to OPA instances. On the OPA end, you can also configure the Discovery feature, which instructs OPA instances to automatically download new discovery bundles when appropriate.

Phase 5 Deploy!

1. ***Deploy policy to OPA.*** It's time to deliver your policy-as-code to your OPA instances.

2. ***Deploy / refresh data to OPA.*** Define how frequently OPA should refresh its data. For instance, if OPA is making admin access control decisions, should OPA check to see who is on pager duty once every ten seconds, ten minutes or once per day?

3. ***Deploy common libraries to all OPAs that depend on that library.*** Ensure that every OPA instance has the information it needs to make good decisions. For instance, if OPA is checking whether containers are PCI-compliant, does it have access to that data that says whether or not they are?

Phase 6 Monitor and Orchestrate

1. ***Ensure that OPA is managed just like any other service in your organization, for example, you could use orchestration to ensure:***

    a. OPA health. Are your OPA instances alive and acting as expected?

    b. Policy version. Are your OPA instances up-do-date?

    c. Data version. Is the data that feeds OPA up-do-date?

Phase 7 Log

1. ***Record decisions for audit.*** For security and compliance, you will need to prove that OPA did what it was supposed to do, in ways that non-technical professionals can understand. Ensure that you funnel OPA decision logs into security and compliance pipeline tools and hold onto that information for an appropriate length of time, whether that's days, months or years.

Explore the OPA ecosystem and well-documented use cases

As a domain-agnostic building block with an ecosystem of more than 30 off-the-shelf integrations, OPA has many use cases with many tools across cloud-native stack. Some examples include:

+ Kubernetes Admission Control
+ Microservices or service mesh authorization (application and service-level) with tools like Envoy, Kong and Istio
+ Infrastructure-as-a-service validation with tools like Terraform
+ CI/CD pipeline policy enforcement with tools like Spinnaker
+ Docker or Linux authorization
+ Access control for Kafka topics
+ Database and data filtering with Elasticsearch or SQL databases
+ Edge policy enforcement via WASM for CloudFlare Workers
+ Security controls for SSH and sudo, or anything that uses the HTTP API
+ And many more

Use Case: Atlassian

Atlassian is the provider of numerous popular cloud products, such as Jira, Trello, Confluence and BitBucket. To provide these products at scale, Atlassian has built and now hosts more than 1,000 services distributed around the world. As is the experience of many companies, authorization was not initially viewed as platform concern. The inevitable consequence was that many of the services implemented their own authorization mechanisms — resulting in services that were individually security, but an environment that was difficult to control and audit. With the help of OPA, Atlassian built a global authorization platform that made it possible to centrally manage and audit authorization decisions for its global network of services. With OPA in place, Atlassian could continue to scale its services worldwide while ensuring robust security for the enterprise and its millions of customers.

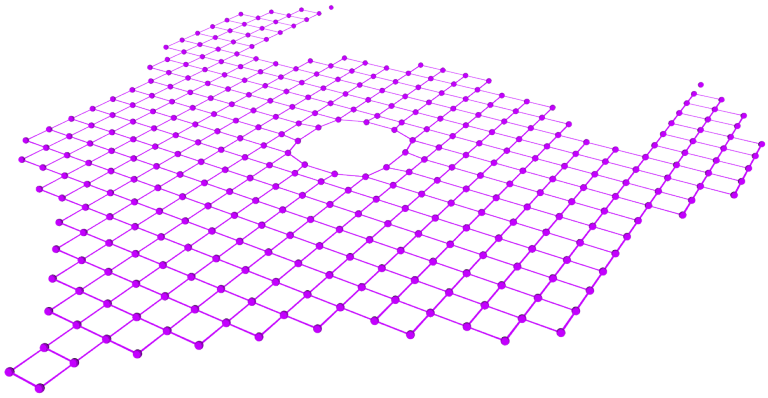
Benefit to the company

It is essential for any cloud-native company to be able to unify and port tools, processes and skill sets across every cloud-native environment. By scaling a standard building block like OPA, companies can deploy consistent and well-validated security and authorization policies across every team, cluster and cloud. This is critical for not only partnering seamlessly witty internal teams, but an ecosystem of integration vendors that support your cloud-native services. Moreover, when security policy and authorization is automated, compliance audits and change validations to your environment become as simple as running a script.

OPA at scale: empowering the enterprise to take advantage of the speed, security and value of policy-as-code

The cloud-native stack is built of abstractions used to manage other abstractions. At a certain point, organizations that deploy OPA at scale, across teams or environments, have a need to adopt higher-level strategies to better operationalize and manage their policy operations. The complexity of the modern stack is compounded when you consider that the teams that manage that stack use different languages, tooling and logic to manage each individual part. With so many custom, hand-built, one-off policies in place, scale becomes an exercise in maintenance instead of innovation.

OPA provides a single, unified language and tooling framework, to eliminate the overhead that stems from building and maintaining custom policy silos. This standardization empowers teams to eliminate the burden of rolling their own policy languages and engines, and provides a seamless way to collaborate across teams. However, even with a standardized policy language, teams still have to manage policy distribution, revisions and impact analysis. As with any tool, scale requires infrastructure to mitigate overhead, rework and one-off manual effort.



The end goal of a tool like OPA is to free developers to focus on business-critical and differentiated problems and building better apps. After all, application development already happens at an unprecedented speed and scale. Yet, as with any distributed solution, it can become time-consuming to ensure all your OPA policy evaluation points are effectively managed, validated, maintained, distributed and monitored in production.

It is worthwhile to consider that, while OPA was built for evaluating and enforcing rules consistently across the stack — and does so with aplomb — it was not designed, on its own, to manage the entire policy lifecycle. For instance, OPA alone is not concerned with:

+ Distributing and managing policy across numerous environments

+ Providing UIs for authoring and constructing policies

+ Aggregating and reporting comprehensive telemetry on OPA decisions

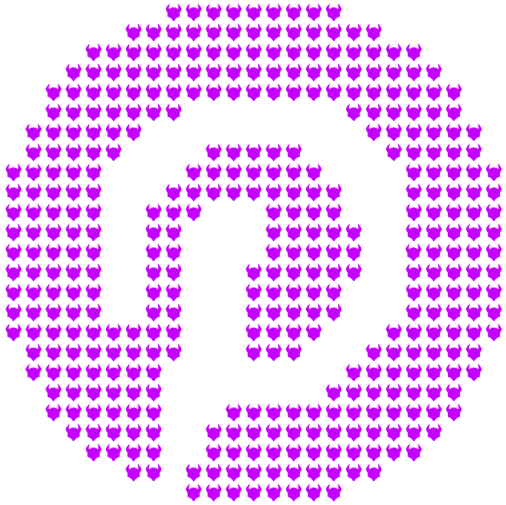
+ Monitoring environments to spot and fix policy violations

+ Validating policy changes against historical data

As a result, managing OPA at scale can likewise become a familiar (if not much more consistent and secure) game of whack a mole. Ultimately, the goal of OPA is still massively powerful, on its own; but in light of these factors, there will be a number of considerations organizations make in order to more seamlessly and securely manage OPA at scale.

OPA policy distribution

At a certain maturity point during every rollout — complexity of environment, level of business-criticality, level of dependency between teams and so on — it simply makes sense to leverage a control or management plane for OPA policy management and distribution. Typically, this is driven by the need to move engineers on to other projects, and/or security and compliance imperatives. As OPA provides APIs that enable control and visibility over policy enforcement, it's possible to build a custom system for policy management and distribution.



Some very large early OPA adopters, such as Netflix and Pinterest, indeed have opted to fashion their own bespoke OPA control planes. This provides companies with useful functionality that fits their specific environment; however, it's a significant and time-intensive undertaking to build a custom control plane, and of course requires resources to maintain it, once deployed. Many OPA adopters find themselves in a build vs. buy situation, when confronted with designing and maintaining a custom solution vs. opting for a more standardized approach.

Integrating OPA with GitOps

Cloud-native organizations are increasingly moving to more mature GitOps workflows, as they provide a robust framework for continuous delivery. To integrate OPA with GitOps, organizations will need a way to store and maintain their policy-as-code in the Git repository, as a single source of truth. This also means, in practice, developers will need 1) an ability to validate any policy changes before committing or deploying them and 2) the ability to automatically fetch policy bundles from Git and distribute them to the correct OPA instances across clusters, clouds and teams.

Simplifying policy building

Rather than custom code OPA policy in Rego every time, most organizations and teams look for simpler ways to build, test and deploy policies that can be reused across teams, across clusters or across clouds. As such, it can be helpful — moving beyond policy templates and macros — to build shared policy libraries that any team, at any OPA skill level, can use. This not only eases the burden of policy creation, but accelerates the continued adoption (and automation) of policy building in the enterprise.

For one example, it can become manually intensive to write JSON schema for less OPA-savvy developers to follow — and then have to translate the file Rego to run tests in OPA, and then report back the results to developers.

It would be better to instead avail those developers with a shared, tested and approved library of OPA policies and tests to increase deployment speed, limit the risk of human error and unload the burden of interpretation and validation from on staff OPA experts.

Pre-runtime validation

Because any change to policy and authorization can result in broken apps or access problems in critical environments, OPA users need to review and validate changes before runtime. This includes things like compliance checks, unit testing and policy output validation. Safeguards like GitOps peer reviews can help to minimize any errors here, but as human beings, developers sometimes make mistakes or get overburdened — particularly when policy is run at scale across dozens of services or millions of instances. As a result, OPA users need to find automated ways to test their policies and validate changes before committing them. One best practice is to keep a record of all historical OPA policy decisions and automatically test policy changes against them — ensuring that future changes act as predicted, helping to minimize security and compliance risk.

Comprehensive monitoring and reporting

When scaled across the organization, OPA policies naturally become critical for compliance, security and privacy. As such, effective monitoring and reporting becomes mission-mandatory. As such, companies should be able to automatically feed anomalous behavior, high-risk rule violations or outages from across your environment directly into DevSecOps processes and SIEMs to enable rapid response.

At the same time, reporting also becomes critical for communicating the value and effectiveness of your policy-as-code systems to other teams, decision makers and non-coding peers. You should be able to provide detailed decision logs — ideally compiled into clear visuals — to demonstrate this value, and prove where controls have been deployed to increase security or maintain compliance. In the same vein, it can also be helpful to show where critical policy implementations or updates have taken effect, along with the value of those implementations. In other words, you should be able to provide the value of your efforts to security and compliance teams, auditors and decision makers — audiences who typically are not coders, and thus should not be expected to read through your policy-as-code implementation.

Efficient policy distribution across environments

Gaining consensus for just one policy can take a substantial amount of cross-functional coordination between numerous teams. As such, it can become challenging to maintain policy consistency across numerous teams, clusters, clouds and more. As OPA scales, you will need to simplify and streamline how you distribute policies across those environments. Oftentimes, this takes shape as a formal process or mechanism used to instill OPA policies consistently into numerous other processes across your organization.

The Business Benefits of Policy Standardization

The wider the adoption of OPA in your organization, the most cost-effective it becomes. When enterprises democratize policy and authorization with a consistent standard, they can improve security and compliance, accelerate application development and increase automation, enterprise-wide.

With OPA running at scale, organizations can:

+ Distribute security and compliance best practices to every team, cluster and cloud across the enterprise

+ Manage policies globally, while enabling complete visibility into every authorization decision across the cloud-native stack

+ Eliminate policy silos, along with the need to hard-code policy logic direction into services

+ Protect against intrusions, prevent lateral movement and hot-patch policies globally to thwart attacks in real time

+ Integrate policy-as-code with established GitOps processes

Use Case: SugarCRM

SugarCRM, a leading customer experience solutions provider with millions of users, needed modern infrastructure security and compliance controls that worked with software-defined systems and could be tracked to prove compliance.

Although SugarCRM used significant automation, the DevOps/platform team still relied on time-intensive manual processes that resulted in manual errors and unnecessary costs (such as unneeded external load balancers). The company was already using OPA, but they knew that to properly scale out their security and compliance guardrails, they needed to establish governance around OPA, while codifying policy and automating best practices. SugarCRM decided to use Styra DAS as a unified control plane for OPA, allowing the company to operationalize OPA policy in production across numerous clusters and automate compliance-as-code in the CI/CD pipeline. As a result of operationalizing OPA, SugarCRM was able to eliminate load balancer costs and risks with policy guardrails, reduce time spent identifying issues and applying standards, simplify audits and reporting with policy-as-code and minimize human errors, security gaps and downtime.

Conclusion

As you can see, there is a great opportunity to manage Open Policy Agent at scale and best support your business, but there are critical elements that need to be in place to future-proof your authorization and policy. Here's a quick recap:

Phase 1: Requirements	*Choose domain*	*Assemble real-world policy*	*Understand data dependencies*	*Choose enforcement points*
Phase 2: Enter OPA	*Configure your services to use OPA.*	*Deploy OPA.*	*Configure OPA.*	*Harden OPA configuration.*
Phase 3: Author Policy	*Write and version control Rego.*	*Decide and learn input and decision schemas.*	*Modularize policy and delegate for collaboration.*	
Phase 4: CI Stage	*Assemble policies from different repositories.*	*Test policies.*	*Create policy and build artifacts (e.g. Bundles).*	
Phase 5: Deploy!	*Deploy policy to OPA.*	*Deploy/refresh data to OPA.*	*Deploy common libraries to all OPAs that depend on that library.*	
Phase 6: Monitor & Orchestrate	*Ensure that OPA is managed just like any other service in your organization.*			
Phase 7: Log	*Record decisions for audit.*			

Styra is the creator of Open Policy Agent (OPA), the open source project dedicated to Cloud-native authorization. OPA was created to address the exponential complexity brought about by today's cloud-native application environments. Legacy approaches to authorization can't scale to meet CICD/DevOps requirements, and can't handle cloud-dynamism or today's privacy needs because they're only focused on people, roles, and accounts.

OPA extends authorization beyond people and accounts to infrastructure and code, allowing enterprises to protect applications as well as the software-defined platforms they run on. Styra Declarative Authorization Service (DAS) is the fastest and easiest way to operationalize OPA at scale across Kubernetes, microservices or custom APIs. Developers, DevOps, and Platform Engineering teams have proven Styra solutions in production to mitigate risk, reduce human error, and accelerate application development in today's dynamic multi-cloud world.

Styra DAS delivers security-as-code for cloud-native. Operational, security, and compliance teams rely on Styra DAS and OPA to quickly create and automatically enforce policy across dynamic multi-cloud environments. Teams can build security, compliance, and operational guardrails directly into their infrastructure—to eliminate errors, and close security gaps. Through its open-source and commercial products, Styra provides the management, intelligence and governance to implement and understand policy across the new stack.

